Chapter 2

Ovis

2.1 Introduction

The work described in this thesis was performed within the framework of the NWO Priority Programme *Language and Speech Technology*. In this chapter I will first introduce the programme to the reader. I will sketch the structure of the project. Some of the choices that lead to this structure had consequences for the work that needed to be done on the natural language understanding module of the project. I will zoom in on this module in the remainder of this chapter. I will describe in some detail the work done by the group responsible for the grammar based natural language proccesing module. A general overview of this work is given and special attention is given to the disambiguation problem.

2.1.1 The goals

The NWO Priority Programme *Language and Speech Technology* is a research programme aiming at the development of spoken language information systems. Its practical goal is to develop a demonstrator of a system that supplies public transport timetable information. Such a system must be capable of recognizing and understanding spoken language. It must furthermore be capable of conducting a natural dialogue in order to obtain the information necessary for answering the user's questions. Finally, the system must be able to produce spoken natural language in order to conduct this dialogue and to produce the answer to the user's request for information. At the time this research programme started (in 1995), such a system did not exist for Dutch. Even though similar systems were developed (or worked on) for other languages (the best known example is probably the ATIS project (**?**; **?**; **?**)), many research questions remained unanswered.

One of the reasons to start the OVIS project was to look at some of the open research questions. In order to meet the practical goal as described above, scientific contributions were envisaged in speech recognition, natural language processing (NLP)and dialogue management. One of the more important scientific goals of this project was to combine insights of linguistics and phonetics on the one hand and computer science on the other. Before the project started it

was expected that the technological requirements of the proposed demonstration version would necessitate a closer study of language 'performance' rather than language 'competence'. It was foreseen that therefore information-theoretic approaches (such as probabilistic techniques) should be investigated. Chapters 4 and 5 will investigate one particular way of incorporating these two approaches in natural language processing.

Apart from the scientific goals, two cultural goals were identified. Firstly, the integration of two different research paradigms, by confronting the (at that time) experimental 'corpus-based' approach with the 'knowledge-based' approaches in linguistic research and language technology. Two separate research groups were working in competition on the NLP module. The ultimate goal was not so much to prove which research paradigm is the most suitable for the task but to form a basis where insights from both research paradigms will lead to cross fertilization. The second cultural goal was to maintain the status of Dutch as a cultural language. It is expected that the status of languages with a small population of speakers (like Dutch) will decline if spoken language technology for that language is not developed to the same extent as for English and German. The NWO Priority Programme 'Language and Speech Technology' was initiated by the Dutch foundation for scientific research (NWO) and co-funded by two industrial partners: Philips Corporate Research and KPN Research. The programme supports engineers from industry to bridge the gap between scientific research and work on commercial Dutch information systems. The cooperation between the Dutch academic community and industrial partners is meant to establish this economic goal of the programme.

When I talk about the OVIS system in this thesis I refer to what is officialy called OVIS2 . OVIS2 is a modularized version of the OVIS prototype. The OVIS prototype is a version of a German system developed by Philips Dialogue Systems in Aachen (**?**; Aust et al., 1995), adapted to Dutch.

2.2 The project set-up

2.2.1 Interfaces

Four academic research groups are involved in the project. The four groups are responsible for and work more or less independent of each other on the three distinct modules: speech recognition, linguistic processing and a combined module for dialogue management and language generation. The research on speech recognition is carried out by the university of Nijmegen (**?**), on dialogue management and language generation by the Institute for Perception Research (IPO) in Eindhoven (**?**; **?**) and, as mentioned above, the work on the NLP module is carried out by two groups who work in parallel (or competition, as you like). A corpus based approach is investigated by the group at the university of Amsterdam (**?**; **?**) and the group in Groningen adopted a grammar based approach. All the work described in the remainder of this thesis is carried out in the context of the grammar based NLP component.

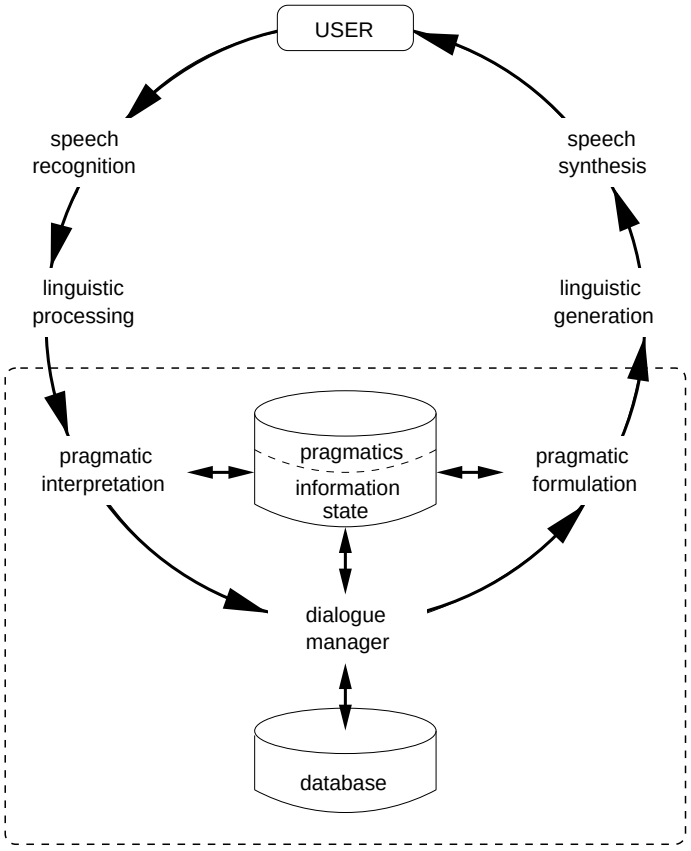

Figure 2.1: The architecture of OVIS2

The fact that on the one hand there is the practical goal of the project, namely the demonstration version, and on the other hand the wish to work as independently as possible on the distinct modules (due to the fact that the research groups are on different locations), we had to agree on the division of labour and on the interfaces bewteen the modules. The architecture of the OVIS2 prototype is given in figure 2.1. In this overview of the architecture, the arrows indicate the flow of information. Boxes are modules of the system. The idea is that the user produces an utterance which is input for the speech recognition module. The result of speech recognition, a set of word hypotheses, is passed on to the linguistic processing component. The result of linguistic analysis, a syntactic analysis and meaning representation, is input for the pragmatic component, which fills in anaphoric references, etc. and passes on the data to the dialogue manager. The dialogue manager checks whether enough information is available to consult the database, or whether further information needs to be requested from the user. Such requests and the answer to the user's query are articulated by means of the synthesis modules. The circled boxes in the center of the figure illustrate the datastructures that are maintained by the different modules. For the motivation behind the chosen structure, the reader is refered to **?**) and **?**).

2.3 The NLP module

All the work described in this thesis is carried out within the box 'linguistic processing'. In this chapter I will focus on the complete NLP module. I have mentioned before that much of the work that needed to be done for this module depends on the work of the other groups. This means that it is crucial to define the interfaces between the separate modules carefully. In the figure defining the architecture of OVIS2 it can be seen that there are two interfaces important for the NLP module.

2.3.1 Interfaces

The input of the NLP component is received from the speech recognizer. The output of the speech recognizer is a *wordgraph* (c.f. section 2.5.1). A wordgraph is a compact representation of all different hypotheses for the spoken input from the user. Linguistic analysis of such a structure is more complicated than analysis of a single input sentence (like, for example, written input), because of the added uncertainty. Not only the linguistic structure of the utterance must be determined, but also which sequence of words (or which one of the available hypotheses in the wordgraph) is closest to the actually uttered sentence. The structure of the wordgraphs is fixed and is explained in section 2.5.1. What is important for the work in this thesis is the fact that from inside the NLP module we do not have any influence on the wordgraph we get from the speech recognizer. We can not return any information to the speech recognizer that influences its behaviour. If, for example, we think that linguistic clues might influence the probability of certain words we can not pass this information on to the speech module, but we have to work with the wordgraph as we get it

from the speech recognizer and express a preference for certain words inside the NLP module.

The interface between the linguistic analysis component and the pragmatic interpretation module is described in (**?**). This interface is defined by a special language, called the *Update Language* (**?**). *Updates* are expressions that modify the information state that is kept in the pragmatics module of the system[1]. At the first stage, the output of the linguistic proccesing is a domain independent semantic representation of the user utterance. This formula is translated into a domain dependent *update expression*. In this way we can keep our grammar as domain independent as possible, while we are still able to make domain dependent decisions. More about the update language and some examples will be provided later (e.g. in section 2.4.8).

2.3.2 Problem Description

While building the linguistic processing module for this spoken dialogue system we have to deal with a number of problems. Firstly, there is the *ambiguity* problem that every system working with natural language has to deal with. The combination of lexical and structural ambiguities often leads to an enormous number of possible readings for an input sentence. In a spoken language system using *wordgraphs* as input, the ambiguity problem is even more acute because of the uncertainty of the input. Before the problem can be solved of which reading of the utterance is the most approriate a choice has to be made between competing hypotheses existing within the wordgraph.

The second problem has to do with the ability of the system to find an analysis in the input, even when the input is not perfectly well formed. This is the problem of *robustness*. Three different kinds of robustness problems can be identified:

- Coverage of the grammar. It is not feasible to anticipate all linguistic constructions that might occur. This is the traditional problem for grammar-based NLP systems. It is clear that it is not desirable that the system fails completely when an unknown linguistic construction is used.
- Spoken language artifacts. Spoken language adds the problem of hesitations, false starts, corrections, etc. It is not always easy to detect these in the input, let alone to correct them.
- Limitations of speech recognition. Even though speech recognition gets better every day and especially for domains with a limited vocabulary the quality of the output of speech recognizers is really good, there are still errors to be expected. The fact that we have to deal with speech over telephone lines makes it even harder in the case of this project.

[1] Although figure 2.1 shows several boxes for the pragmatic processing, all the boxes inside the dotted lines can be considered one module.

Many of the issues discussed in this chapter are directly related to the robustness and ambiguity problems mentioned above. While solving these problems a third issue can not be forgotten: the *efficiency* of the system. The practical goal of the project is to construct a demonstrator. Given the nature of the proposed application, it will be clear that the system is supposed to run in *real time*. This means that the user of the system can expect to get a reply within reasonable time limits. This requirement provides a further challenge for the NLP component.

2.3.3 Overview

In the first part of this chapter I will give an overview of how the grammar is set up. I will not give a full account of the syntactic coverage of the grammar, nor will I explain in too much detail why certain decisions were made. For a full description of the grammar I refer to **?**) or for a shorter version to **?**).

The second part of the chapter consists of a description of the robust parsing algorithm that is used. Again in this part, I will not try to give a complete account of the parsing algorithms. The purpose of both these two sections is to give the reader an idea of how the grammar and parser are set up, why it is set up like this (accounting for the work done during the first two years of the project) and most importantly, I want to provide a basis for the extensions I propose in chapters 3 and 5. The former requires an understanding of the syntax and an extension of the semantics of the grammar. The latter makes extensive use of the decisions taken during parser construction.

The third section is about evaluation. In this section I will explain how evaluation criteria are defined and how well the OVIS2 grammar performs both in terms of accuracy and efficiency. I will use these results for comparison in later chapters were I discuss my extensions. The final section of this chapter is an evaluation of the errors made by the OVIS2 grammar and discusses how much we could gain by taking dialogue context as an extra information source into account.

2.4 A computational grammar for Dutch

During the first two years of the OVIS project a grammar for the OVIS spoken dialogue system was developed. We started off with a corpus of human/human dialogues that were recorded using a system that had similar functionality. Although human/human dialogues can be very different from human/machine dialogues, it gave us an impression of the kind of utterances we could expect in our system. One of the goals we set for the project was to develop a grammar that on the one hand meets the requirements for this application (i.e. fast processing and robustness are important issues) and on the other hand would be the basis for a general computational grammar for Dutch. Even though utterances in human/human dialogues are often more elaborate than the ones we would face in our system, using these dialogues helped us to move forward

towards the general grammar. When the OVIS prototype became operational, examples of human/machine dialogues became available. We used a bootstrapping approach for improving our grammar by testing the grammar with data provided by the prototype. New, improved versions of the prototype became available on a regular basis, giving us fresh (and more acurate) data to test and improve our grammar.

In developing the OVIS2 grammar we have tried to combine the short-term goal of developing a grammar which meets the requirements imposed by the application (i.e., robust processing of the output of the speech recognizer, extensive coverage of locative phrases and temporal expressions, and the construction of fine-grained semantic representations) with the long-term goal of developing a general, computational, grammar which covers all the major constructions of Dutch.
The design and organization of the grammar, as well as many aspects of the particular grammatical analyses we propose, are based on Head-driven Phrase Structure Grammar (HPSG) (**?**). We depart from the HPSG formalism mostly for computational reasons. As is explained below, the grammar is compiled into a restricted kind of definite clause grammar (DCG (**?**)) for which efficient processing is feasible. The semantic component follows the approach to monotonic semantic interpretation using *quasi-logical forms* presented originally in Alshawi (1992).
The grammar currently covers the majority of verbal subcategorization types (intransitives, transitives, verbs selecting a PP, and modal and auxiliary verbs), NP-syntax (including pre- and postnominal modification, with the exception of relative clauses), PP-syntax, the distribution of VP-modifiers, various clausal types (declaratives, yes/no and WH-questions, and subordinate clauses), all temporal expressions and locative phrases relevant to the domain, and various typical spoken-language constructs. Due to restrictions imposed by the speech recognizer, the lexicon is relatively small (2500 word forms, many of which are names of stations and cities).

2.4.1 Formalism

The formalism that we use for the OVIS2 Grammar is a variant of Definite Clause Grammar (DCG) (**?**). We have chosen for DCG because:

- DCG provides for a balance between *computational efficiency* on the one hand and *linguistic expressiveness* on the other.
- DCG is a (simple) member of the class of declarative and constraint-based grammar formalisms. Such formalisms are widely used in linguistic descriptions for NLP.
- DCG is straightforwardly related to context-free grammar. Almost all parsing technology is developed for CFG; extending this technology to DCG is usually possible (although there are many non-trivial problems as well).

We impose a few extra requirements on the OVIS2 grammar formalism. I only mention the two most important restrictions. First, we require that rules can easily be mapped on their 'context free skeleton'. We will later see how this works. The major implication is that (unlike 'pure' HPSG) we can not use underspecification of categories in rules. And further, to allow head-driven parse strategies, we require that for every rule the head daughter is specified. An efficient head-corner parsing strategy for this formalism is discussed in **?**).

Grammar rules. Grammar rules are defined by the predicate `rule/3`. An example is given in:

(2.1)
```
rule(vp_vpnp, vp(Subj,Agr,Sem),
        [v(Subj,Agr,trans,l(Arg,Sem)),np(_,Arg)]).
```

The first argument is the rule identifier (must be a ground prolog term). The second argument specifies the mother category (the predicate name) and the feature values that are shared with daughters (the arguments). The third argument is a (non-empty) list of daughter constituents. All daughters consist of a category name and a number of arguments. In order to satisfy the requirement that a rule can be mapped on its context free backbone, it is necessary here that the categories of mother and daughter constituents are non-variable prolog terms and that the length of the list of daughters is known. In this particular rule, the head verb passes 'Subj' (subject) and 'Agr' (agreement) information to the mother, the verb is required to be 'trans' (transitive) and the semantics is given by a lambda term in which the semantics of the NP is fed as an argument (see lexical entry example 2.2).
Terminal symbols cannot be introduced in rules directly, but are introduced by means of lexical entries.

Lexical entries. The lexicon is defined by the predicate `lex/2`. As an example, the lexical entry 'sleeps' could be encoded as:

(2.2)
```
lex(sleeps,v(np,agr(3,sg),intrans,l(X,sleep(X)))).
```

The first argument is the terminal symbol introduced by this lexical category. It is allowed to be a (non-empty) list of atoms when a lexical entry introduces a sequence of words (e.g. *Centraal station* (central station)). The second argument is the category (a non-variable term).

Feature constraints. Almost all work in computational grammar writing uses *feature structures* of some sort. It is fairly standard to compile (descriptions of) such feature structures into first order terms (see **?**) for a recent overview). For the OVIS grammar, the HDRUG development platform (**?**) is used. It contains a library that implements the compilation of feature constraints into Prolog terms, and various predicates to visualize such Prolog terms as feature structures in matrix notation.

I will often write rules as the example I gave above (2.1) in matrix notation, as follows:

(2.3) `rule( vp_vpnp,` $\begin{bmatrix} vp \\ \text{subj} & \boxed{1} \\ \text{agr} & \boxed{2} \\ \text{sem} & \boxed{3} \end{bmatrix}$, $\begin{bmatrix} v \\ \text{subj} & \boxed{1} \\ \text{agr} & \boxed{2} \\ \text{vform} & \text{trans} \\ \text{sem} & \lambda(\boxed{4},\boxed{3}) \end{bmatrix}$, $\langle \begin{bmatrix} np \\ \text{agr} \\ \text{sem} & \boxed{4} \end{bmatrix} \rangle$).

The following grammar fragment is a typical example of how grammar rules are specified:

(2.4)
```
rule(1,S,[Np,Vp]) :-
        S => s, np(Np), vp(Vp),
        Vp:vform => finite,
        subj_agreement(Vp,Np).

    np(Np) :- Np => np, Np:lex => -.
    vp(Vp) :- Vp => v,  Vp:lex => -.
    subj_agreement(Vp,Np) :- Vp:agr <=> Np:agr.
```

In this fragment the assignment operator ('=>') is used to assign types to structures (e.g. 'S => s' means that structure 'S' must be of type 's' and '$Np : lex$ => $-$' means that the value of attribute 'lex' of 'Np' must be of type '-'). The path operator (':') is used to point at attributes inside a structure. And finally the equality operator ('<=>') indicates that two attributes share the same value.

2.4.2 Signs

In unification-based grammar formalisms, linguistic information is represented by means of typed feature-structures. Each word or phrase in the grammar is associated with such a feature-structure, in which syntactic and semantic information is bundled. Within Head-driven Phrase Structure Grammar (HPSG), such feature-structures are called *signs*, a terminology which will be followed here.

At present, the grammar makes use of 15 different types of sign, where each type roughly corresponds to a different category in traditional linguistic terminology. For each type of sign, a number of features are defined. For example, for the type NP (noun phrase), the features AGR (agreement), NFORM (noun form), CASE, and SEM (semantics) are defined. These features are used to encode the agreement properties of an NP, (morphological) form, case and semantics, respectively.

There are a number of features which occur in most types of sign, and which play a special role in the grammar. The feature SC (SUBCATEGORIZATION) (present on signs of type v (verb), *sbar* (dependent clause), *det* (determiner), a (adjective), n (noun) and p (preposition), for instance, is a feature whose value is a list of signs. It represents the subcategorization properties of a given sign. It is used to implement rules which perform functor-argument application (as in Categorial Grammar).

The feature SEM is present on all signs. It is used to encode the semantics of a word or phrase, encoded as a *quasi logical form*. The feature MOD is present on the types *a* (adjective), *pp* (prepositional phrase), *p* (preposition), *adv* (adverb), *sbar* and *modifier*. It is used to account for the semantics of modifiers. Its value is a list of *quasi-logical forms*. The details of the semantic construction rules and principles are dealt with in section 2.4.7.

2.4.3 Syntax: principles, structures, and rules

Although the grammar formalism we have chosen is based on HPSG, it is because of the *context free backbone requirement* imposed by the grammar-parser interface, that (unlike HPSG) we can not rely on general rule schemata completely. In order to stay as close as possible to the generality of the rule schemata, we have adopted an approach in which we classify the *grammar rules* in various *structures*, which are in turn defined in terms of general *principles*.

In definition 2.5 I give an other example of the definition of a rule. The `np-det-n` rule introduces a *head-complement* structure in which (following the traditional semantic analysis) the determiner is the head, and the noun the complement:

(2.5)
```
rule(np_det_n, NP, [Det, N]) :-
        NP => np, Det => det, N => n,
        NP:nform => norm,
        hd_comp_struct(Det,[N],NP).
```

In the first line of the definition the requirement that the categories of the constituents are known is fulfilled. The second line adds the constraint that *nform* of the resulting NP is `norm`[2]. The last line classifies the `np-det-n` rule as a *head complement* structure, whose head constituent is the *determiner* who needs just one *complement* to form an NP.

Definition 2.6 says that a *head complement structure* is a *headed structure*, and in 2.7 we can see that a *headed structure* inherits from the *head feature principle* (which states the mother constituent and the head daughter must agree on certain features) . The HEAD FEATURE PRINCIPLE states for a number of features (the *head-features*) that their value on the head daughter and mother must be unified. As this principle generalizes over various types of sign, its definition requires the predicate `unify_ifdef`[3].

(2.6)
```
hd_comp_struct(Head,Complements,Mother) :-
        hd_struct(Head,Complements,[],Head,Mother).
```

[2]The attribute *nform* was introduced in this grammar in order to distinguish between 'normal' NP's and *temporal* and *locative* NP's. In order to give the latter a special treatment in certain situations (which sometimes is desirable in this domain), we need a distinguishing feature.

[3]`unify_ifdef` checks if a particular feature is defined for the head daughter and the mother constituent; it will only unify the values if this is the case.

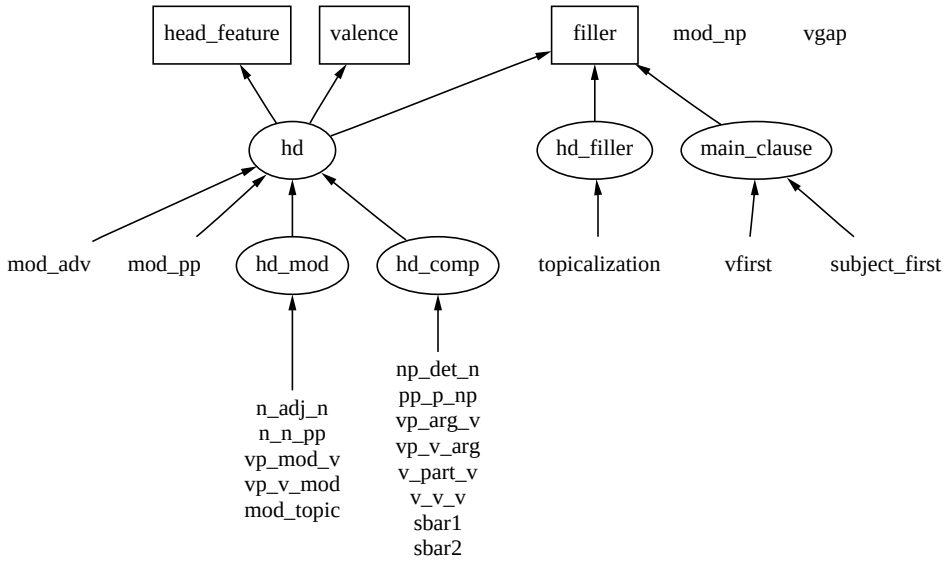

Figure 2.2: The Rule Hierarchy (with PRINCIPLES shown in boxes, *structures* in ovals, and `rules` without frame). Note that the `mod_np` rule (a unary rule which transforms temporal NPs into verbal modifiers) and the `vgap` rule (a rule which introduces verbal gaps) are exceptional in that they do not inherit from general principles.

(2.7)
```
hd_struct(Head,Complements,SemanticHead,Mother) :-
        head_feature_principle(Head,Mother),
        valence_principle(Head,Complements,Mother),
        filler_principle(Head,[],Mother),
        SemanticHead:sem <=> Mother:sem.
```

(2.8)
```
head_feature_principle(Head,Mother) :-
        unify_ifdef(Head,Mother,vform),
        unify_ifdef(Head,Mother,agr),
        unify_ifdef(Head,Mother,case),
        unify_ifdef(Head,Mother,mod),
        unify_ifdef(Head,Mother,pform),
        unify_ifdef(Head,Mother,aform),
        unify_ifdef(Head,Mother,vslash),
        unify_ifdef(Head,Mother,subj).
```

An overview of all grammar rules defined in the fragment at the moment, together with the structures and principles from which they inherit, is given in figure 2.2.

2.4.4 The lexicon

The lexicon is a list of clauses `lex(Word,Sign)`, associating a word (or sequence of words) with a specific sign.
Constraint-based grammars in general, and lexicalist constraint-based grammars in particular, tend to store lots of grammatical information in the lexicon.

```
intransitive(Pred,Sign) :- iv(Sign), iv_sem(Sign,Pred).

transitive(Pred,Sign) :-   tv(Sign), tv_sem(Sign,Pred).

v(V) :- V => v, V:lex => basic,
    V:vslash => [], V:subj <=> [Subj],
    Subj => np, Subj:nform => norm.

iv(V) :- v(V), V:sc <=> [].

tv(V) :- v(V), V:sc <=> [Obj],
    Obj => np, Obj:nform => norm, Obj:case => acc.
```

Figure 2.3: Fragment of the lexical hierarchy

This is also true for the OVIS2 grammar. A lexical entry for a transitive verb, for instance, not only contains information about the morphological form of this verb, but also contains the features SC[4] and SUBJ[5] for which quite detailed constraints may be defined. Furthermore, for all lexical signs it is the case that their semantics is represented by means of a feature-structure. This structure can also be quite complex. To avoid massive reduplication of identical information in the lexicon, the use of inheritance is essential.

In figure 2.3, the use of inheritance in the lexicon is illustrated. All lexical entries for verbs have a number of properties in common, such as the fact that they are of type *v* (verb), and take a normal (non-locative and non-temporal) NP as subject. This is expressed by the template `v(V)`.[6] Intransitive verbs (`iv(V)`) can now be characterized syntactically as verbs which do not subcategorize for any (non-subject) complements. Transitive verbs (`tv(V)`) subcategorize for an NP with accusative case. The templates `intransitive(Pred,Sign)` and `transitive(Pred,Sign)`, finally, combine the syntactic and semantic properties of intransitive and transitive verbs. The variable `Pred` is used in the semantics to fix the value of the predicate defined by a particular verb.[7] The

[4] SC: 'subcategorized-for' elements. These are the objects and indirect objects that the verb governs.

[5] SUBJ: 'subject'. The properties the subject of the verb must have.

[6] The restriction that subjects must be [NFORM *norm*] must be overwritten for verbs which select a dummy subject (such as weather verbs) and 'raising' verbs which do not impose a semantic (or thematic) restriction on their subject.

[7] By default, the value of `Pred` for a given word is identical to the root form of that word.

attribute-value matrices for the templates `iv(V)` and `tv(V)` are:

(2.9) `iv(`
$$\begin{bmatrix} v \\ \text{lex} & \text{basic} \\ \text{sc} & \langle\rangle \\ \text{subj} & \left\langle \begin{bmatrix} np \\ \text{nform} & \text{norm} \end{bmatrix} \right\rangle \\ \text{vslash} & \langle\rangle \end{bmatrix}$$
`).` `tv(`
$$\begin{bmatrix} v \\ \text{lex} & \text{basic} \\ \text{sc} & \left\langle \begin{bmatrix} np \\ \text{nform} & \text{norm} \\ \text{case} & \text{acc} \end{bmatrix} \right\rangle \\ \text{subj} & \left\langle \begin{bmatrix} np \\ \text{nform} & \text{norm} \end{bmatrix} \right\rangle \\ \text{vslash} & \langle\rangle \end{bmatrix}$$
`).`

The lexicon itself (i.e. the predicate `lex/2`) is defined in terms of the predicates `entry`, `inflection` and `lexical_rules`:

(2.10)
```
lex(Word,Sign) :-
    entry(Root,Sign0),
    inflection(Root,Word,Sign0,Sign1),
    lexical_rules(Sign1,Sign).
```

The specification of `entry(Root, Sign)` defines for each root form what its associated sign is. For instance, for verbs we must typically distinguish a first person singular form, a second and third person singular form, and a plural form (which is also the form of the infinitive). The predicate `inflection` defines how inflected forms are derived. For example, there is an inflection rule which adds a `t` to the base form of a verb, and specifies that its agreement features are third person singular, and its VFORM (verb form) value is *fin*. Lexical rules can be used to transform the sign associated with a lexical entry. For instance, the account of nonlocal dependencies makes use of a lexical rule which removes a sign from SC and places it on SLASH. A more detailed account of this lexical rule is given in the section on nonlocal dependencies in (**?**).[8] As an example, assume the stem `arriveer` (to arrive) is defined as an intransitive:

(2.11)
```
entry(arriveer,Sign):-
    intransitive(arriveren,Sign).
```

Such a definition will give rise to a number of lexical entries. One of these will be the third person singular finite form:

(2.12) lex(arriveert,
$$\begin{bmatrix} v \\ \text{lex} & \text{basic} \\ \text{vform} & \text{fin} \\ \text{sc} & \langle\rangle \\ \text{subj} & \left\langle \begin{bmatrix} np \\ \text{nform} & \text{norm} \\ \text{agr} & \text{sg} \wedge \text{3rd} \end{bmatrix} \right\rangle \\ \text{vslash} & \langle\rangle \end{bmatrix}$$
).

[8] Such lexical rules are 'compiled-out' during compilation time. Although we are sympathetic to the account in (**?**), the use of delayed evaluation techniques turned out to be too expensive for the purpose of this grammar.

2.4.5 Syntactic coverage

Although one of the goals we set for the development of the OVIS grammar was to create the basis for a general computational grammar for Dutch, we also had to meet the requirements for this particular project. The limited domain of answering questions about public transport time tables, did not require the coverage of some syntactic constructions that are otherwise used frequently in Dutch. The fact that added complexity of a grammar might mean on the one hand more (possibly spurious) ambiguity and the other hand longer processing times, suggests that it is better to limit the coverage of the grammar to those linguistic constructions that are actually used. This implies not only that the coverage in the lexical domain is limited, but also that several grammatical constructions are not taken into consideration at all or accounted for only to a certain extent. For example, passive constructions are hardly used in these dialogues and therefore not covered by our grammar. Dutch verb clusters can be quite complex, but the main (and most frequently used) constructions can be dealt with with a limited number of rules. However, the fact that certain linguistic constructions are excluded does not mean that they can not be covered in principle. Most of the gaps can simply be filled in by adding additional rules. The practical goal mentioned earlier just requires a trade off between syntactic coverage and computational complexity. In section 2.6 the evaluation of the NLP module is discussed. As part of the evaluation, not only wordgraph input is tested, but also the transcribed test-sentences. When evaluating the grammar on a corpus of 1000 (previously unseen) transcribed test-sentences, a semantic concept accuracy of 95% is obtained (**?**). Given the fact that we have to deal with spoken language (which has several problems with respect to grammaticality, as mentioned before), we think that the coverage of the grammar is quite satisfactory.

The fact that the second (long term) goal of creating a basis for a general computational grammar for Dutch is also met, is currently proven by the development of such a grammar within the recently started project *Algorithms for Linguistic Processing* (**?**) at the university of Groningen. The *Alpino* grammar developped for this project (**?**) is an extension of the OVIS grammar. It is, just like the OVIS grammar, a lexicalized grammar in the tradition of HPSG, but with this grammar it is proven that it is possible to design a grammar formalism that is not only efficient and robust, but allows linguistically sophisticated analyses as well.

For an extensive description of the syntactic coverage of the OVIS grammar, the reader is refered to section 2.5 of **?**). Here I confine myself to an overview of the included grammar rules.

Noun phrases

The noun phrases (NP's) are covered by the following three rules:

```
np --> det n
n --> adj n
n --> n pps
```

The only unusual feature defined for constituents of type *n* is the feature *nform* (noun form). This feature can have the values *norm* (normal), *loc* (locative), *temp* (temporal) and *num* (numeral). It appeared to be very useful for this domain because of the importance (and special treatment) of especially locative and temporal noun phrases.

Prepositional phrases

Two grammar rules are needed to cover prepositional phrases:

```
pp --> p np
pp --> p np part
```

Where the second rule covers the cases where a particle of a verb follows a PP, as in: *ik wil naar Assen toe* ('I want to go to Assen').

Verb phrases

In the following rule schemata variable category names (ARG and MOD) are used. This is just for notational convenience. Both ARG (arguments) and MOD (modifiers) can be instantiated with several constituents. Rewrite rules for ARG and MOD are defined in the OVIS grammar. There are two rule schemata for combining a verb with an argument:

```
vp --> ARG v
vp --> v ARG
```

Noun phrase complements will be found left of the verbal head: *een kaartje kopen* ('to buy a ticket'). Prepositional phrase complements may either precede or follow the verbal head: *vanuit Assen vertrekken, vertrekken vanuit Assen* ('depart from Assen').

A verb modifier can preceed or follow the verb:

```
vp --> MOD v
vp --> v MOD
```

A modifier can either be an adverb (*Ik wil onmiddelijk vertrekken* ('I want to leave immediately')), a prepositional phrase (*[Ik moet] om twee uur in Assen zijn / in Assen zijn om twee uur / om twee uur zijn in Assen* ('I have to be in Assen at two o'clock')) or a temporal NP (*[Ik moet] morgen naar Assen / naar Assen morgen* ('I have to go to Assen tomorrow')). Three unary rules are

defined to map these categories to signs of type *modifier*.

The particle might be detached from the verb (e.g. *(dat ik om twee uur) aan wil komen* ('that I want to arrive at two o'clock')):

```
v --> part v
```

The following rule allows simple verb clusters where the modal verb precedes its infinitive complement (e.g. *(dat ik om tien uur) wil vertrekken* ('that I want to leave at ten o'clock')):

```
v --> v v
```

Subordinate clauses

Subordinate clauses (e.g. *dat de trein naar Groningen gaat* ('that the train is going to Groningen')) are covered by the following rule:

```
sbar --> comp subj vp
```

Main clauses

Main clauses with a finite verb in intial position (as in yes/no-questions; e.g. Wilt u nog een andere verbinding? ('Would you like another connection') are of type *ques*:

```
ques --> vmain subj vp
```

Main clauses in which the finite verb appears in second position (as in declarative sentences (e.g. *Rob koopt geen kaartje* ('Rob does not buy a ticket') or WH-questions) (e.g. *Wie koopt een kaartje?* ('Who buys a ticket?')) are of type *root*):

```
root --> subj vmain vp
```

Wh-questions and topicalisation

The first phrase in a main clause can also be a (non-subject) complement or modifier. The nexty two rules cover these cases. First, sentences with a fronted complement (e.g. *Naar welk station wilt u reizen?* ('To which station would you like to travel?'):

```
root --> topic ques
```

And finally sentences starting with a modifier (e.g. *Hoe laat gaat de volgende trein naar Zwolle?* ('When does the next train to Zwolle leave?')

```
root --> mod ques
```

2.4.6 Special grammar rules

The domain which has been selected for OVIS (information dialogues concerning public transportation) and the fact that OVIS deals with spoken language, imply that it is crucial that a number of grammatical phenomena are described in a robust manner. In particular, temporal expressions, locative expressions (names of cities and stations), and a number of typical spoken language constructions, such as greetings, occur frequently in such dialogues.

The grammar rules and lexical entries for these phenomena make use of the OVIS2 grammar formalism, but are not organized according to the linguistic principles discussed above. This is true not only for the syntax, but also for semantics. The reason for dealing with these phenomena by means of a set of more or less *ad hoc* rules and lexical entries is that the constructions discussed below are often extremely idiosyncratic. At the same time, describing the regularities that can be observed does not seem to require the overhead of the grammar architecture we assume for the rest of the grammar. The most economical and robust solution seemed therefore to encapsulate the grammar for these constructions in relatively independent grammar modules.

2.4.7 Semantics

In this section I explain in some detail how semantics is dealt with in the OVIS grammar. Semantics is presented in more detail since the extension proposed in chapter 3) builds on it and therefore presuposes detailed knowledge.

The output of the grammatical analysis is a semantic, linguistically motivated and domain-independent representation of the utterance in the form of a Quasi Logical Form (QLF). The QLF formalism was developed in the framework of the *Core Language Engine* (CLE, (Alshawi, 1992)). Since then, the formalism was used and further developed in projects such as the *Spoken Language Translator* (**?**), *Clare* (**?**), in the *Fracas*-project (**?**) and in *Trace & Unification Grammar* (**?**). In OVIS the QLF is translated into a domain-specific *update* expression, which is passed on to the pragmatic interpretation module and dialogue manager for further processing. The dialogue manager maintains an information state to keep track of the information provided by the user. An *update* expression is an instruction for updating the information state. Below, the choice for QLFs as semantic representation language is motivated and I will discuss how these QLFs are translated into updates.

The semantic representation language

Predicate logic is often used to represent the meaning of sentences. Due to its long tradition in describing semantics of natural languages it is now a well established and well understood technique. Nevertheless, it often seems helpful to extend the predicate calculus with more advanced techniques, such as *generalized quantifiers* or *discourse markers*.
The main advantage of artificial languages like predicate logic is that they are unambiguous. An ambiguous natural language utterance will therefore correspond to more than one expression in predicate logic, one for each reading of the utterance. The disadvantage of this approach is that for very ambiguous inputs, expensive computations must be carried out to compute all readings. The alternative adopted in *monotonic semantic interpretation* based formalisms (**?**) is to represent ambiguity by means of underspecification and to postpone the computation of all possible readings (See also **?**) and **?**)).
In QLF many ambiguities can be expressed by underspecification. Furthermore, it turns out that the computation of the domain-specific meaning of an utterance can be carried out without resolving all the sources of ambiguity in the QLF for that utterance. For instance, the scope of quantifiers seems to be to a large extent irrelevant for the computation of *update* expressions, and thus a representation in which quantifier-scope is undetermined is sufficient for our purposes.
For example, sentences involving a negation operator and a quantifier (as in 2.13) are ambiguous with respect to the scope of these operators. In this domain, however, we know that we do not want to consider the second reading (2.13c). Underspecification of the quantifier scope gives us the opportunity to express our preference by translating the QLF directly (i.e. without calculating all the possible scopings) into the desired *update*.

(2.13)

a. Gaat er niet een latere (trein)?
   Is there not a later train?

b. $not(\exists x \text{latere\_trein}(x) \wedge \text{gaat}(x))$

c. $\exists x \text{latere\_trein}(x) \wedge not(\text{gaat}(x))$

Quasi logical form

Events are handled in the tradition of Davidson (**?**). Verbs introduce a new (existentially quantified) event variable. When modifiers are attached to a verb, the translation of this modifier is conjoined with the translation of the verb. Both conjuncts share the same event-variable. The translation of *De trein vertrekt om vier uur* (The train leaves at four o'clock) will be:

(2.14)
```
and(vertrekken(exist(e,event(e)),de(x,trein(x))),
      at(e,hour(4)))
```

In figure 2.4 the QLF produced by the current grammar for the example sentence *Ik wil om ongeveer vier uur vertrekken* (I want to leave at about four o'clock)

is given. Note that the arguments of the predicates *willen* and *vertrekken* are (generalized) quantifiers that are unscoped with respect to each other.

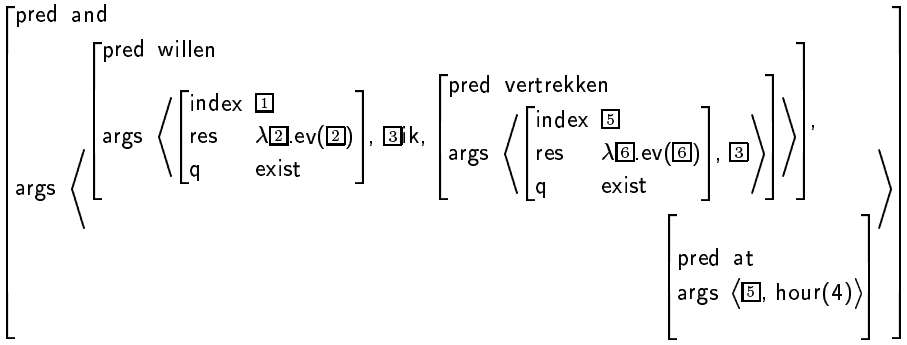

Figure 2.4: QLF for the utterance 'Ik wil om ongeveer vier uur vertrekken'

Sentences like the one in figure 2.4 are ambiguous with respect to the attachment of the prepositional phrase. The PP *om vier uur* can either modify *willen* or *vertrekken*.

The implementation of QLFs

Our implementation of QLF in the OVIS grammar follows roughly the presentation in (?). Because QLFs can be translated in updates directly, it will not be necessary to resolve the QLFs completely and therefore some of the apparatus supplied by the original formalism can be omited.
A QLF is either a *qlf-term* or a *qlf-formula*. A *qlf-term* is one of the following:

- a term index[9]
- a constant term
- an expression of the form: `t_expr(Index,Cat,Restr,Quant)` [10]
  in which `Cat` is a list of feature-value equations, denoting some extra-linguistic information. At this moment it includes only the features `lex` (the lexical entry associated with this term) and `type` (e.g. quantifier, pronoun, etc.); `Restr` is a predicate logical expression (possibly with lambda-abstraction); `Quant` is a generalized quantifier/determiner.

A QLF formula is one of the following:[11]

- the application of a predicate to arguments:
  `p_form(Predicate,[Arg1,...,Argn])` in which *Predicate* is a higher order predicate (including logical operators like `and`, `not`, etc). Arguments may be formulas or terms.

[9] In the original formalism *indices* and *variables* are distinguished. An index uniquely identifies a `term` expression. At this moment indices and variables have the same function in our implementation. We may need to distinguish between them later.

[10] In chapter 5 of (?) a fifth argument *Referent* is used. This argument may specify a contextual restriction on the range of quantification.

[11] In chapter 5 of (?) two more formula constructs are introduced. These are not used in the current implementation.

Every man
qlf $\texttt{t\_expr}(\texttt{i}, \langle \texttt{lex} = \texttt{alle}, \texttt{type} = \texttt{q} \rangle, \lambda \texttt{x}.\texttt{man}(\texttt{x}), \forall)$
pl $\forall x.(man(x) \rightarrow ....)$

yawns
qlf $\texttt{v\_form}(\texttt{x}, \texttt{s\_form}(\texttt{Scope}, \texttt{p\_form}(\texttt{yawns}(\texttt{t\_expr}(\texttt{e}, \langle \texttt{type} = \texttt{event} \rangle, \lambda \texttt{y}.\texttt{event}(\texttt{y}), \exists), \texttt{x}))))$
pl $\lambda x.\exists y.(event(y) \wedge yawn(y, x))$

Every man yawns
qlf $\texttt{s\_form}(\texttt{Scope}, \texttt{p\_form}(\texttt{yawn}(\texttt{t\_expr}(\texttt{e}, ...), \texttt{t\_expr}(\texttt{i}, ...))))$
pl $\forall x.(man(x) \rightarrow (\exists y.event(y) \wedge yawns(y, x)))$
$\exists y.(event(y) \wedge \forall x.(man(x) \rightarrow yawns(y, x)))$

Figure 2.5: The relation between an expression in QLF and a fomula of predicate logic

- a predicate logical formula with lambda-abstraction: `v_form(Var,Formula)`. This type does not exist in the original formalism. This type is added to access variables that are deep down in the structure.
- a formula with scoping constraints: `s_form(Scope,Formula)` in which `Scope` is either a variable when scoping is underspecified or a list of term-indices occurring in `Formula`, in the order in which they are scoped. For example, if index `i` has scope over index `j`, then `Scope=[i,j]`.

The definitions can best be illustrated with a simple example in which a QLF expression with its corresponding formula in predicate logic is compared. In figure 2.5 the sentence *Every man yawns* is given both a translation in QLF and in Predicate Logic. In the translation of the full sentence it can be seen that nothing is said about the scope of respectively the quantifier of the event associated with the verb and the quantifier of its argument. All the information that is needed to resolve the scoping relation is stored in the arguments (of type `t_expr`) of *yawn*. Resolving the scoping relation would instantiate the variable `Scope` to `[i,e]` (for every man there exists a moment that he yawns) or to `[e,i]` (there exists a moment that every man yawns).

Construction of QLFs

During grammatical analysis QLFs are constructed compositionally. In *head complement structures* the head daughter is the syntactic as well as the semantic head of the structure. This means that the semantic content of the complement constituent is combined with the semantic content of the head. The value of the SEM feature of the head is passed up to the mother.
In *head modifier structures* the modifier is the semantic head. The semantics of the syntactic head of the structure is plugged into the MOD feature of the modifier. Below I will show how the semantics of the modifier is combined with

the semantics of the constituent it modifies. The value of the SEM feature of the modifier is passed up to the mother.

Determiners. Consider the lexical entry for *alle* (*all*) in figure 2.6. The definition for a determiner will require that this constraint holds between the sign and the quantifier introduced by the particular determiner, in this case ALL. Determiners subcategorize for a noun. The semantics of that noun specifies the restriction of the determiner.

Nouns. The semantic content of the noun *trein* (*train*) is given in figure 2.6. Note that it is assumed that quantifiers may scope at this point. The value of the `scope` feature is left unspecified here.

Adjectives. *Adjectives* are modifiers. They operate on structures whose semantic content is of type `v_form`. The lambda variables of the two formulas are unified and the semantic content of the structure is the conjunction of the logical formula of the adjective and the logical formula of the structure it modifies. An example (*eerste* (*first*)) is given in figure 2.6.

Verbs. The semantic content of the subject with which the *intransitive verb* combines, is stored in the SUBJ feature. This information is plugged into the second position of the argument list of the verb. The verb introduces a new event. The first argument of the argument list of the verb is filled with the term representing the event. The index associated with this event is unified with the variable of V_FORM at the highest level, to make the event variable much easier to access. An example (*vertrekken* (*to leave*)) is given in figure 2.7.
The *transitive verb* is defined similar to the intransitive verb. The only extra information we have to take care of here is the second argument of the verb. The second argument can be found on subcat (i.e the third element on the verb's argument list must be unified with the value of the SEM feature of the element on SC).
Consider the example of a `modal_verb` in figure 2.7. The subject argument of the *modal verb* is taken from the SEM feature of SUBJ. The index, which uniquely identifies this term, is unified with the value of SEM of the subject argument of the VP-complement. The VP-complement argument is unified with the value of SEM of the V on the SC list. The modal verb and the vp-complement refer to the same event, by unifying the event variable of the modal verb with the event variable of the vp-complement.

Verbal modifiers. The semantics of *adverbs* resembles that of *adjectives*. The semantic constraints for the preposition *op* (*on*) heading a prepositional phrase which acts as a verbal modifier is given as an example in figure 2.6.
The semantics consists of a conjunction of the formula of the constituent it modifies and an extra constraint on this formula. The constraint is headed by the preposition and its arguments are the lambda variable of the formula it constrains and the formula of the NP it subcategorizes for.

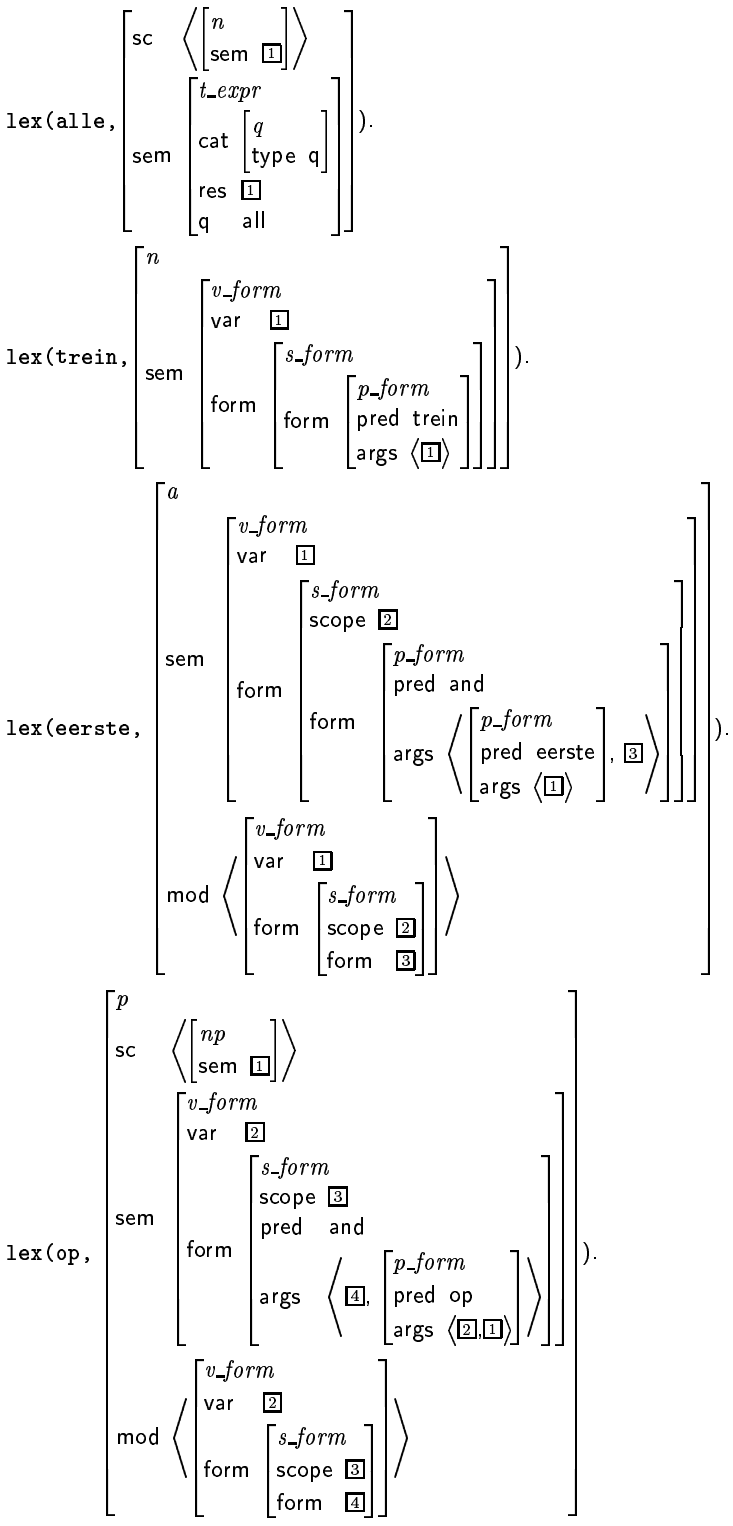

Figure 2.6: Examples showing the semantics of determiners, nouns, adjectives and prepositions.

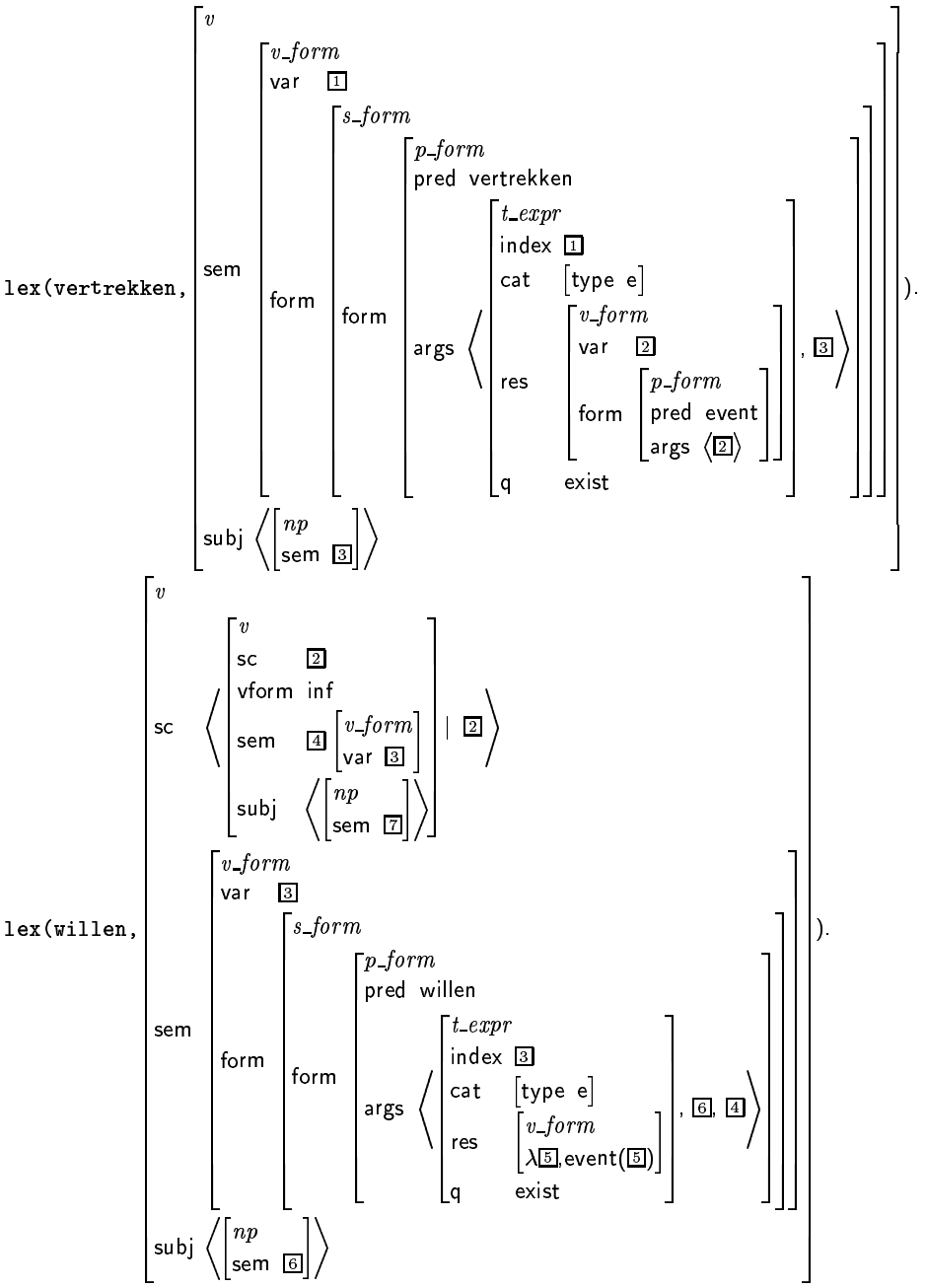

Figure 2.7: Examples showing the semantics of intransitive and modal verbs.

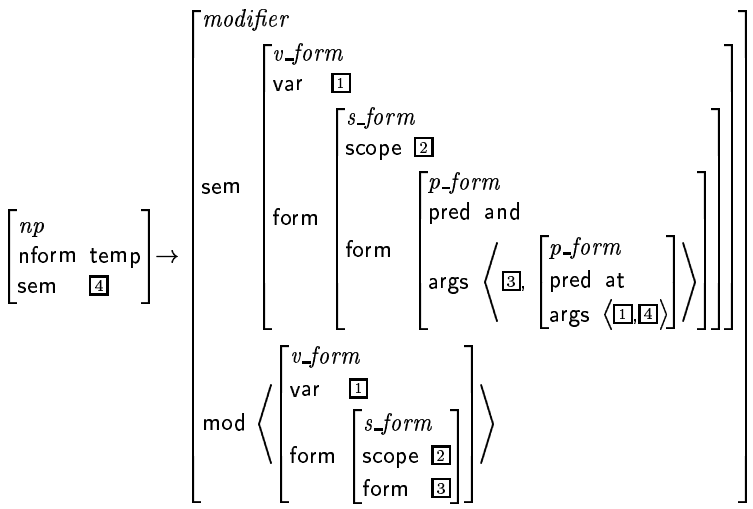

Figure 2.8: Rule `mod_np` to treat temporal noun phrases as modifiers.

In Dutch, temporal NPs can act as verbal modifiers:

(2.15) a Ik wil *zondag* vertrekken
I want to leave on sunday

b Ik wil *drie januari* naar Amsterdam
I want to go to Amsterdam on the third of January

c Ik wil er uiterlijk *drie uur* zijn
I want to arrive at three o'clock

As NPs normally do not have a modifier semantics, there is a unary rule that transforms temporal NPs into modifiers (figure 2.8). The structure that is modified is specified in the MOD feature. The semantic content of the modifier is constructed as if it was a PP with P_FORM *om* (at). The semantic content of the (temporal) NP daughter is plugged into the second position of the argument list of the preposition.

2.4.8 Constructing updates from QLFs

QLFs are translated into *updates* to be passed on to the dialogue manager for further processing. The dialogue manager keeps track of the information provided by the user by maintaining an *information state* or *form*. This form is a hierarchical structure, with slots and values for the origin and destination of a connection, for the time at which the user wants to arrive or depart, etc. The distinction between slots and values can be regarded as a special case of ground and focus distinction (Vallduvi, 1990). Updates specify the ground and focus of the user utterances. For example, the utterance *"No, I do not want to travel to Leiden but to Abcoude!"* yields the following update:

(2.16)
```
user.wants.travel.destination.
                        ([# place.town.leiden];
                         [! place.town.abcoude])
```

An important property of this representation is that it allows encoding of speech-act information. The "#" in the update means that the information between the square brackets (representing the focus of the user-utterance) must be retracted, while the "!" denotes the corrected information.
The effect of the update 2.16 is illustrated by the following two information states. If 2.16 is applied to the upper information state in 2.9, the result is the lower state.
As I will explain in the following section, the result of parsing will be a sequence of phrasal projections. Each of these is associated with a QLF. The assumption is that these QLFs are domain independent. The following translation procedure constructs a domain dependent *update* expression on the basis of such a sequence. This procedure consists of two steps.
The first step of the procedure comprises the application of a number of rules which transform a given subsequence of QLFs into some other QLF, based on domain-specific assumptions. A typical example is the rule which translates the QLFs corresponding to a sequence of two locative noun phrases:

(1) Amsterdam, Groningen

In this case, the sequence of two QLFs is mapped to a single QLF which can be paraphrased as *to travel from Amsterdam to Groningen.*
As another example, the adverb *graag* is translated into a QLF indicating *confirmation* if the adverb occurs in isolation. Such a translation is motivated by dialogues of the following type:

(2.17) a. Dus U wilt van Amsterdam naar Groningen?
So you want to travel from Amsterdam to Groningen?

b. Graag.
Please.

The second step of the translation procedure actually compiles an update expression on the basis of the resulting sequence of QLFs. By recursively descending the given QLFs a number of rules is applied which translate parts of QLFs into partial updates. Basic QLF constructs are translated into their equivalents in the update language. For example the QLF part associated with *'om vier uur'* ('at four o'clock') in (fig 2.4) is translated into:

(2.18) `moment.at.time.clock_hour.4`

These rules are often constrained by specific requirements on the context of those QLF parts. Elaborating on the example just given, *'om vier uur vertrekken'* ('to leave at four o'clock') will be translated in:

(2.19) `origin.moment.at.time.clock_hour.4`

If there are more modifiers attached to *'vertrekken'*, these translations will also be governed by '`origin`': *'Om vier uur uit Assen vertrekken'* ('To leave at four o'clock from Assen')

(2.20)
```
origin.( place.town.assen;
         moment.time.at.clock_hour.4)
```

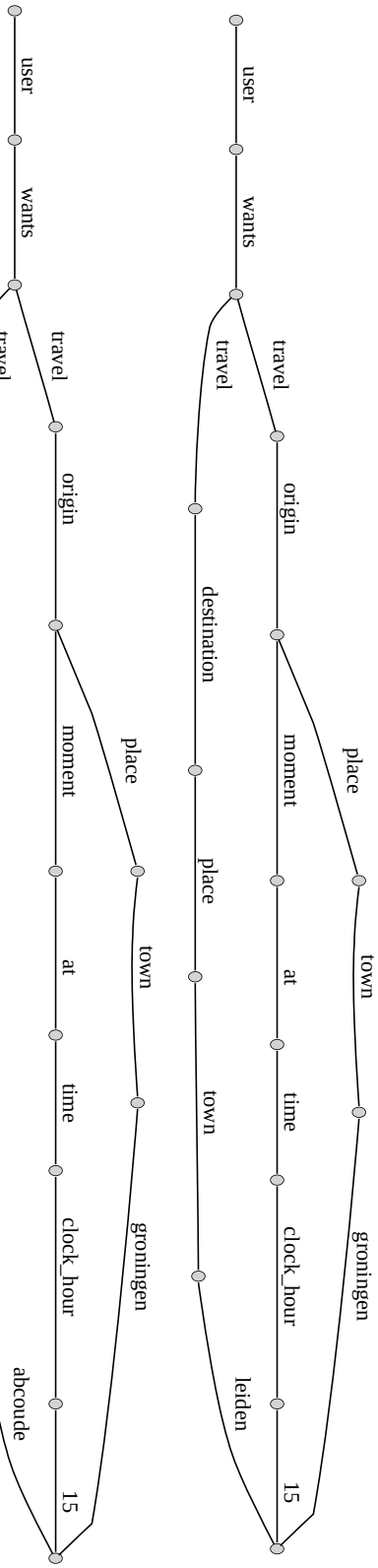

Figure 2.9: Information state before (upper) and after (lower) the update given in 2.16

This strategy gives us the opportunity to give a different translation in different contexts. This can best be demonstrated with the slightly more complicated case of negation in sentences:

- *'Nee'* ('No')
  `[=no]`

- *'Nee, naar Assen'* ('No, to Assen')
  `destination.[!place.assen]`

- *'Niet'* ('No')
  `[=no]`

- *'Niet naar Assen'* ('Not to Assen')
  `destination.[#place.assen]`

Although *'Nee'* and *'Niet'* get the same translation when they occur in isolation, their interpretation is quit different when they occur in context. *'Nee'* gives rise to a correction, while *'Niet'* yields a retraction.

2.5 Robust parsing

2.5.1 Wordgraphs

The speech recognizer passes its output to the NLP module in the form of *word-graphs* (**?**). A word-graph is a compact representation of all the hypotheses for a spoken utterance generated by the speech recognizer. It consists of a number of states that represent points in time and transitions between them. There is one start state that marks the beginning of an utterance and there are one or more final states marking the end of the utterance. The transitions are annotated with words, the words that are hypothesized by the speech recognizer between two points in time, and an *acoustic score* representing a measure of confidence that the hypothesized word was actually uttered. The confidence measures are given as the negative logarithms of their probabilities.

At an early stage, the word-graph is normalized to eliminate the *pause transitions*. Such transitions represent periods of time for which the speech recognizer hypothesizes that no words are uttered. After this optimization, the word-graph contains exactly one start state and one or more final states, associated with a score, representing a measure of confidence that the utterance ends at that point. In figure 2.10 a word-graph as produced by the speech recognizer is given together with its normalized counterpart. From now on, it is assumed that word-graphs are normalized in this sense. Below, I refer to transitions in the word-graph using the notation $trans(v_i, v_j, w, a)$ for a transition from state v_i to v_j with symbol w and acoustic score a. Let $final(v_i, a)$ refer to a final state v_i with acoustic score a.

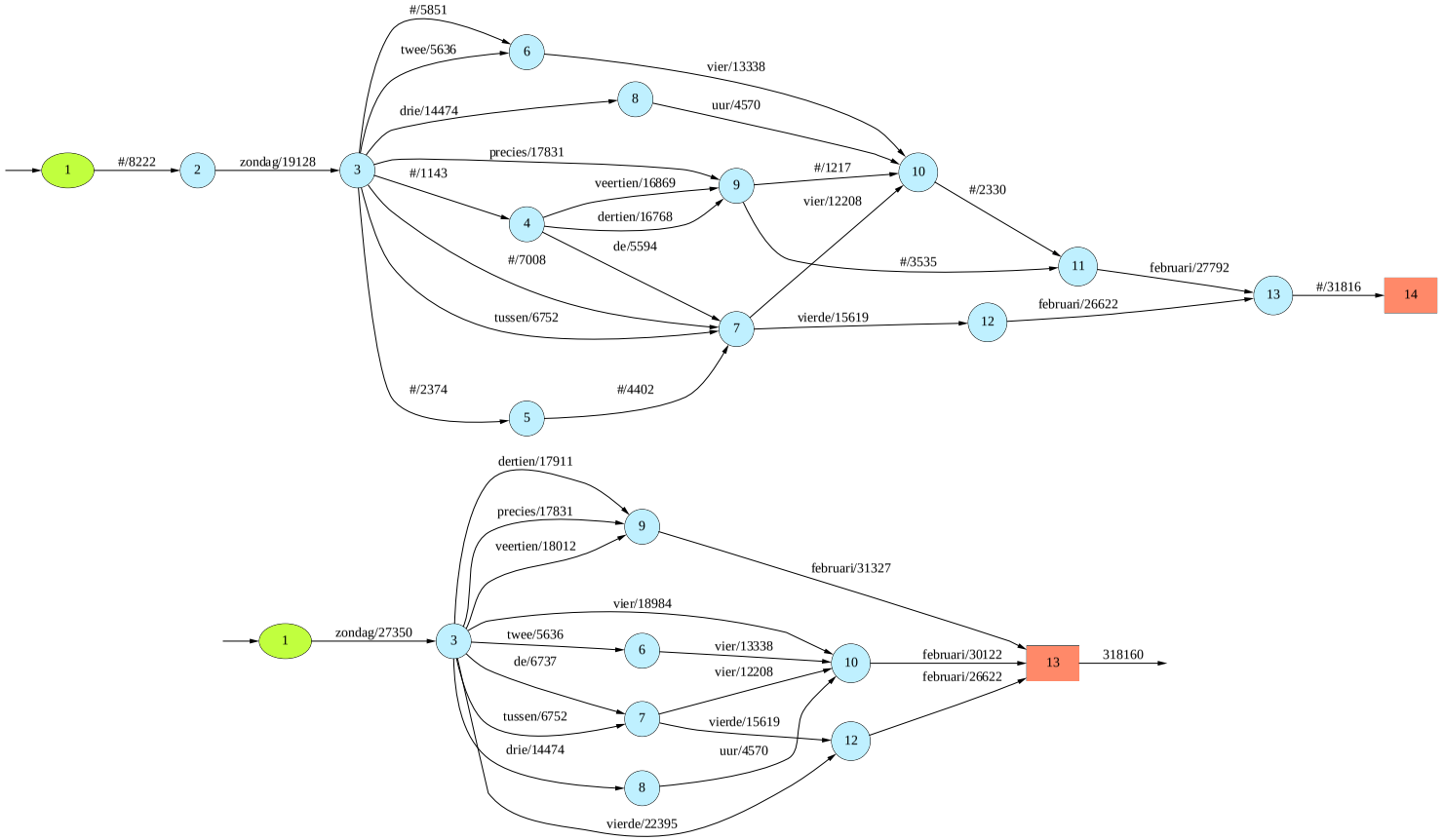

Figure 2.10: Example of a word-graph produced by the speech recognizer and its normalized counterpart. The word-graph was constructed for the utterance *Zondag vier februari* (Sunday Februari fourth). The special label # indicates a pause transition.

2.5.2 Parsing Wordgraphs

In stead of word strings, the input of the parser will be a normalized wordgraph as defined in the previous section. In order to use this input format a parser that can deal with wordgraphs has to be constructed. A straightforward and easy, but computationally infeasible solution would be to list all paths from start node to final node and parse them all. The number of paths grows too quickly with the number of transitions per word in the wordgraph. The number of 22.8 transitions per word reported for the test set described in section 2.6.1 corresponds to several milions paths per wordgraph. Moreover, as many paths wil overlap, a lot of work will be done several times with this approach. In this section a parsing strategy that allows the reuse of parse results of partial paths is presented.

A second problem arises from the fact that we have to deal with spoken input. Spoken language tends to be less well formed than its typed counterpart. Sentences can be ungrammatical, and they can also contain corrections, for example:

(2) Ik wil van Assen, nee van Haren naar Amsterdam
I want from Assen, no from Haren to Amsterdam

or repeated words, like in:

(3) Ik wil van van Assen naar Amsterdam
I want from from Assen to Amsterdam

These examples typically do not make it very hard to understand what is meant, but do create serious problems for a grammatical analysis of the complete utterance. Even though it can be argued that it is a good idea to anticipate repairs as in (2) in a grammar for a spoken dialogue system, it might not be necessary to include grammar rules to cover these cases. It certainly is inadvisable to include rules for the latter case. The fact that the input is provided by a speech recognizer is another potential problem. Even though the performance of speech recognizers is getting very reliable (especially when the lexicon is small), recognition errors still have to be dealt with. The performance can be influenced negatively by for example a noisy environment or a bad telephone line. This might mean that some words of the utterance are not understood correctly and that, as a consequence of this, the actual uttered sentence is not available in the wordgraph. Finally it must be recognized that even for such a small and relatively easy domain as we are dealing with here, it will be impossible to write a grammar that covers every grammatical construction that the user can think of. It is very likely that some utterances, however grammatical, will be missed by the grammar.

In short, three sources of potential problems when parsing wordgraph input have to be dealed with: The actually uttered sentence might

- be available, but is not grammatical (due to irregularities in the spoken utterance)

- not be available (due to speech recognition errors)
- be available and grammatical, but the linguistic construction involved is not covered by the grammar

A spoken dialogue system functioning in the real world will encounter one of the above mentioned situations frequently and can not afford a strategy in which it fails whenever this is the case. The recognizer and language processor must perform robustly in these cases. A strategy is called for that allows one to retrieve all the available information from the wordgraph whenever, for whatever reason, a complete analysis can not be given.

When we look for example at the utterance in (3) it would be good if we were able to retrieve the most informative parts of the utterance: the prepositional phrases *'from Assen'* and *'to Amsterdam'* even if we could not analyze the whole utterance. This is what a 'concept spotting' approach to language analysis in a spoken dialogue system typically would return. A proper linguistic approach should be able to do the same thing whenever it fails to supply a full analysis.

?) propose to search in the input for what are called *maximal projections*. These maximal projections can be analyzed by the grammar as top categories. In the current grammar they can be categories like S (sentence), NP (noun phrase) and PP (prepositional phrase). Even though several constituents can be recognized as top-categories, for the maximal projections there will always be a preference for constituents that cover as many words as possible. Ultimately the complete utterance can be analyzed as a sentence. So, in example (3) the NP *'Amsterdam'* could be analyzed as a top-category, but the PP *'naar Amsterdam'* will be preferred. In order to analyze the whole utterance in (3), the first four words have to be skipped before a sequence of two maximal projections can be found. The advantage is obvious: even when the full analysis can not be given, certain meaningful constituents might be found, preserving the message of the utterance. As said before, in this domain the users tend to answer with short utterances, often only consisting of a (temporal or locative) noun phrase or prepositional phrase. It seems to be justified to acknowledge this fact by allowing the grammar to analyze these constituents as top-categories.

The next step is to find these maximal projections inside the wordgraph. Normally, the parser tries to find instances of the top-categories that stretch from the start state of the wordgraph to a final state. The proposed backing off approach for those cases where an analysis of the complete utterance is not possible means that this strategy must be generalized to finding *all* the top-categories *anywhere in the wordgraph*. Not only on the paths from start state to final state, but also on *all partial paths*.
Thus a parser that finds all major categories anywhere in the wordgraph is required. If a bottom-up chart parser is used, then we might use the inactive chart items for this purpose. However, in the OVIS project it was decided not to be constrained to a particular parsing strategy, which led to a different approach. Section 3.5 of **?**) shows that in a logic programming setting the use of

underspecification of the state names associated with the top-most goal obtains the desired effect, without loss of efficiency.

After the parser has finished, the result is a word-graph annotated with a number of instances of top categories. For each of these categories we are interested in the word-graph state where this category starts (v_i), the word-graph state where this category ends (v_j), the sequence of symbols associated with this category (x), the accumulated acoustic score (a), and the qlf (q). Let $\mathit{parsed}(v_i, v_j, x, a, q)$ refer to such categories.

Later in this chapter I will refer to the definition of an annotated wordgraph, so I repeat the definition as given in **?**). The *annotated word graph* is a directed acyclic graph (V, E, v_s, v_f) where

- V is the set of vertices consisting of the states of the word graph $v_0 \ldots v_n$, and a new vertex v_{n+1}. v_s is the start state of the word graph and the start state of the annotated word graph. $v_f = v_{n+1}$ is the single final state of the annotated word graph.
- E is the set of edges consisting of:
  1. *skip* edges. For all $\mathit{trans}(v_i, v_j, w, a)$ there are edges $(v_i, v_j, w, a, \epsilon)$.
  2. *category* edges. For all $\mathit{parsed}(v_i, v_j, x, a, q)$ there are edges (v_i, v_j, x, a, q).
  3. *stopping* edges. For all final(v_i, a) there are edges $(v_i, v_{n+1}, \epsilon, a, \epsilon)$.

```
 1. parsed(1,   3,        < zondag >,          27350,    sunday       )
 2. parsed(3,   9,        < precies >,         17831,    exactly      )
 3. parsed(3,  10,        < drie,uur >,        19044,    3 o'clock    )
 4. parsed(3,  12,        < de,vierde >,       22356,    the fourth   )
 5. parsed(3,  13,  < de,vierde,februari >,    48978,    february 4   )
 6. parsed(3,  13,   < dertien,februari >,     49238,    february 13  )
 7. parsed(3,  13,   < veertien,februari >,    49239,    february 14  )
 8. parsed(3,  13,     < vier,februari >,      49106,    february 4   )
 9. parsed(6,  13,     < vier,februari >,      43460,    february 4   )
10. parsed(7,  13,     < vier,februari >,      42330,    february 4   )
```

Figure 2.11: Annotated wordgraph: Output of the parser after parsing the (normalized) wordgraph of figure 2.10.

An example of an annotated wordgraph is given in figure 2.11. The input of the parser was the normalized wordgraph of figure 2.10. The items are annotated with the source state, the target state, de sequence of symbols found between source and target state, the accumulated accoustic score and the semantic representation of the sequence of symbols. Normally the semantic representation is a QLF, but for reasons of readability the QLF is replaced here with the English translation of the found sequence of words.

2.5.3 Search the annotated wordgraph

In order to do a search for the best path in the wordgraph, it is first of all necessary to define what available knowledge can be used to distinguish a good path from a bad one. The second thing to worry about is how important the different knowledge sources are (i.e., how much each of them contributes to the total score of a path). Finally an algorithm is needed to search the wordgraph in order to find the optimal path.

Knowledge sources

The weights associated with the vertices in the wordgraph depend on information from several sources. Evidence from the speech recognizer is taken into account, but obviously we also want to exploit the parsing results and last, statistical language models can be very informative:

Speech

- Acoustic score
  The wordgraph input represents all the sequences of words that the user might have said according to the speech recognizer. The evidence for the hypothesised words is translated into an *accoustic score*. Obviously, the acoustic score present in the wordgraph is an important factor. The acoustic scores are derived from probabilities by taking the negative logarithm. For this reason we aim to minimize this score in order to maximize likelihood. If edges are combined, then we have to sum the corresponding acoustic scores.

Linguistic

It is argued before that there is a preference for hypotheses that span as much time as possible between start and final state in the graph. The output of the parser, the annotated wordgraph, supplies the information needed. Two types of clues are available:

- Number of 'skips'
  The goal is to minimize the number of skips in order to obtain a preference for the maximal projections found by the parser. Each time a skip edge is selected, the number of skips is increased by 1.

- Number of maximal projections
  We want to minimize the number of maximal projections, in order to obtain a preference for more extended linguistic analyzes over a series of smaller ones. Each time a category edge is selected, this number is increased by 1.

Statistical

Simple statistical language models have proven to be very useful in language processing. Here we would also like to benefit from this knowledge.

- Ngrams
  We have experimented with bigrams and trigrams[12]. Ngram scores are expressed as negative logarithms of probabilities. This implies that combining Ngram scores requires addition, and that lower scores are better.

Putting the evidence together

Now that there are four different kinds of evidence available (and more sources might be added later), a method for putting the evidence together (and for computing a weight that is associated with a particular vertex) is needed in order to compare the competing hypotheses. The only requirement made to ensure that efficient graph searching algorithms are applicable is that weights are *uniform*. This means that a weight for an edge leaving a vertex v_i is independent of how state v_i was reached.

Weights are expressed as tuples $< c_1, ..., c_n >$, where each c_i is a cost component. At this moment weights are four-tuples $< c_1, c_2, c_3, c_4 >$ where the components $c_1 ... c_4$ stand for 'number of skips', 'number of maximal projections', 'accoustic score' and 'ngram score' respectively. For each cost component an *initial value* is specified and each edge in the graph specifies a value for each cost component. An *update function* is defined to modify the value of each of the cost components when the path is extended. This *update function* takes a multi-dimensional weight (i.e. the tuple $c_1 ... c_n >$) and an edge as arguments and returns the modified multi-dimensional weight.

Finally, an ordering on these multi-dimensional weights must be defined. The function *total* is defined as follows:

$$(2.21) total(\langle c_1, c_2, c_3, c_4 \rangle) = c_4 + k_{nlp} * (c_1 + c_2) + k_{wg} * c_3.$$

where, k_{wg} and k_{nlp} are constants denoting the importance of the constribution of the 'accoustic score' and of both the linguistic cost components respectively.

I would especially like to note here the complication induced by the *ngram* component because of the uniformity requirement mentioned above. The problem is caused by the fact that the graph does not have a memory. So, in a particular state only those words which are associated with the edges *leaving* this state can be seen. In order to compute a *trigram* probability (i.e. the probability of a word given the previous two words: $P(w_0|w_{-2}, w_{-1})$) we need some information about the path we followed so far. Just adding some kind of memory does not solve the problem, because *uniformity* here requires that every state of the graph is independent of how this state was reached. The solution suggested in **?**) and **?**) is to alter the definition of the graph in such a way that the relevant $n-1$ words are part of the vertex. This results in a graph where each vertex contains information about the word graph state *and* about the $n-1$ labels

[12] An introduction to *ngram* models is given in section 4.1.2

that were seen to reach this state. For a full explanation (and the formal definition of the annotated wordgraph) the reader is reffered to the above mentioned sources.

Search algorithm

The problem of finding the optimal path in the annotated wordgraph can be characterized as a search problem. The adopted search algorithm is a variant of the DAG-SHORTEST-PATH algorithm (**?**). This algorithm finds shortest paths for uniformly weighted directed acyclic graphs. I will not explain the algorithm as used by the OVIS parser here, but refer again to **?**).

2.6 Evaluation

In this section I will explain how the NLP module of the OVIS2 system was evaluated and give an idea of how well the grammar was performing about three years after the project started. The conclusion will be that although the results are satisfactory, there were still some cases where some improvement might be possible. In the next section I will follow this up with a discussion of some of these observations and I will suggest a possible way of dealing with them. That section will serve as an introduction to the research described in the remainder of this thesis.
First I will have a look at the available corpora and give an idea of how hard the task was. Then I will say a few words about effiency matters. I will give an overview of how well the NLP module was performing in terms of accuracy in the last part of this section. These accuracy measures will be used for comparison in later chapters.

2.6.1 Corpora

Earlier in this chapter I mentioned that due to the bootstrapping approach, new data sets (of improved versions of the OVIS2 prototype) became available all the time. This had some consequences for comparison of the experiments described in the following chapters.

About three years into the project an evaluation was set up to compare the performace of the competing NLP modules: the 'grammar based' module described here and in more detail in **?**) versus the 'data oriented' (probabilistic, DOP) module described in **?**) and **?**). To compare the two NLP modules, a formal evaluation was carried out on 1000 new, unseen, representative wordgraphs (obtained using the latest version of the speech recognizer). Full details on the evaluation procedure, and all the evaluation results are described in **?**) and **?**). For these wordgraphs, annotations were provided by our project partners, consisting of the actual sentences ('test sentences') and updates ('test updates'). This test set is also used for the experiments described in chapter 5. At the time I ran the experiments of chapter 3 this test set was not yet available. For those experiments a different test set is used. I will say more about this in

chapter 3.

For training of the *ngram* models, a corpus of 60.000 user utterances was available[13]. Also available were the dialogues that these utterances were taken from. So for the experiments of Chapter 5 I had 60K 'system question'/'user utterance' pairs available. The 60K utterances correspond to almost 200K words, giving an average of a bit more than 3 words per uttterance. A characterization of the test-set in terms of length of utterances and size of wordgraphs is given in table 2.1.

	graphs	transitions	words	t/w	w/g	t/g	max(t/g)
			Train				
Input	60000		190000		3.2		
			Test				
Input	1000	48215	3229	14.9	3.2	48.2	793
Normalized	1000	73502	3229	22.8	3.2	73.5	2943

Table 2.1: **Characterization of test set (1).** Note that the training corpus consisted of the utterances only (wordgraphs were available for some of those utterances, but not for all of them)

A further indication of the difficulty of this set of word-graphs is obtained if we look at the word and sentence accuracy obtained by a number of simple methods. The string comparison on which sentence accuracy and word accuracy are based is defined by the minimal number of substitutions, deletions and insertions that is required to turn the first string into the second (Levenshtein distance d (**?**; **?**)). Word accuracy is defined as $1 - \frac{d}{n}$ where n is the length of the actual utterance.
The method *speech* only takes into account the acoustic scores found in the word-graph. The method *possible* assumes that there is an oracle which chooses a path such that it turns out to be the best possible path. This method can be seen as a natural upper bound of what can be achieved. The methods *bigram* (*trigram*) report on a method which *only* uses a bigram (trigram) language model. The methods *speech_bigram* (*speech_trigram*) use a combination of bigram (trigram) statistics and the speech score.

2.6.2 Efficiency

Although *efficiency* is recognized to be crucial for the application we are looking at in the OVIS project, I will not put much emphasis on this aspect in this thesis. I will merely repeat the state-of-the-art at the moment of the evaluation

[13]In table 2.4 I give two figures; One for the results at the time of the official evaluation, the other for the results a year later. One of the reasons for the better results in the latter tests, was the fact that at that moment an additional 40K question/answer pairs was available. The characteristics of these extra 40K utterances were comparable with the first 60K. In chapter 5 I will use the full corpus of about 100K question/answer pairs.

method	WA	SA
speech	69.8	56.0
possible	90.4	83.7
bigram	69.0	57.4
trigram	73.1	61.8
speech_bigram	81.1	73.6
speech_trigram	83.9	76.2

Table 2.2: **Characterization of test set (2).**

input	method	mean msec	max msec	max kbytes
test sentence	grammar-based	28	610	524
word graphs	grammar-based N=2	298	15880	7143
word graphs	grammar-based N=3	1614	690800	34341

Table 2.3: **Efficiency.** N refers to the n-gram size.

mentioned before. In the remainder of the thesis I will just look at the impact certain approaches have on the efficiency of the NLP module. When solutions have an inherently disastrous effect on the efficiency, they should be avoided. Especially because one of the conclusions that could be drawn after the evaluation of the 'grammar-based' NLP module, was that although people tend to be sceptical about the feasibility of parsing full utterances in spoken dialogue systems, sophisticated grammatical analysis is fast enough in this application.
It is argued in **?**) that the average CPU-times are a bit misleading. Especially for parsing the wordgraphs the variance in parse times was quite large. Some wordgraphs are so big that the parser had serious difficulties processing them. A second measurement made clear that when bigrams were used in the module, the parser returned a result within 1000 milliseconds CPU-time in 95% of the cases. When the n-gram size was extended to 3, the figure was about 87 %.

2.6.3 Accuracy

Word accuracy provides a measure for the extent to which linguistic processing contributes to speech recognition. However, since the main task of the linguistic component is to analyze utterances semantically, an equally important measure is *concept accuracy*, i.e. the extent to which semantic analysis corresponds with the meaning of the utterance that was actually produced by the user.

For determining concept accuracy, the test set must be annotated with the desired semantics. Each user response was annotated with an *update* representing the meaning of the utterance that was actually spoken. The annotations were made by our project partners in Amsterdam, in accordance with the guidelines given in **?**).
Updates take the form described in section 2.4.8. An update is a logical for-

Method	String accuracy		Semantic accuracy			
	WA	SA	match	precision	recall	CA
Input: test sentence						
grammar-based	N/A	N/A	95.7	95.7	96.4	95.0
Input: wordgraph						
grammar-based N=2	82.3	75.8	80.9	83.6	84.8	80.9
grammar-based N=3	84.2	76.6	82.0	85.0	86.0	82.6
(1 year later:)						
grammar-based N=3	84.5	77.4				83.7
speech + trigrams	83.9	76.2				83.2

Table 2.4: **Accuracy.** N refers to the n-gram size.

mula which can be evaluated against an information state and which gives rise to a new, updated information state. The most straightforward method for evaluating concept accuracy in this setting is to compare (the normal form of) the update produced by the grammar with (the normal form of) the annotated update. A major obstacle for this approach, however, is the fact that very fine-grained semantic distinctions can be made in the update-language. While these distinctions are relevant semantically (i.e. in certain cases they may lead to slightly different updates of an information state), they often can be ignored by a dialogue manager. For instance, the two updates below are semantically not equivalent, as the ground-focus distinction is slightly different.

(2.22)
```
user.wants.destination.place.([# town.leiden];
                              [! town.abcoude])
user.wants.destination.([# place.town.leiden];
                        [! place.town.abcoude])
```

However, the dialogue manager will decide in both cases that this is a correction of the destination town.

Since semantic analysis is the input for the dialogue manager, concept accuracy is measured in terms of a simplified version of the update language. Inspired by a similar proposal in (Boros et al., 1996), each update is translated into a set of *semantic units*, where a unit (here) is a triple ⟨`CommunicativeFunction, Slot, Value`⟩. For instance, the example above, as well as the example in section 2.4.7, translates as:

(2.23)
```
⟨ denial, destination_town, leiden ⟩
⟨ correction, destination_town, abcoude ⟩
```

Both the updates in the annotated corpus and the updates produced by the system were translated into semantic units of the form given above.

Semantic accuracy is given in the following tables according to four different definitions. Firstly, the proportion of utterances for which the corresponding

semantic units exactly match the semantic units of the annotation (*match*) is listed. Furthermore *precision* (the number of correct semantic units divided by the number of semantic units which were produced) and *recall* (the number of correct semantic units divided by the number of semantic units of the annotation) is calculated. Finally, following (Boros et al., 1996), concept accuracy is presented as

$$CA = 100\left(1 - \frac{SU_S + SU_I + SU_D}{SU}\right)\%$$

where SU is the total number of semantic units in the translated corpus annotation, and SU_S, SU_I, and SU_D are the number of substitutions, insertions, and deletions that are necessary to make the translated grammar update equivalent to the translation of the corpus update.
The results listed in table 2.4 are achieved for the test-set of 1000 word-graphs mentioned in section 2.6.1. String accuracy is presented in terms of word-accuracy (WA) and sentence accuracy (SA). Semantic accuracy includes the percentage of graphs which receive a fully correct analysis in terms of semantic units (match), percentages for precision and recall of semantic units, and concept accuracy as defined by the formula given above.

The first three lines of table 2.4 report on the status of the grammar-based NLP module at the time of the official evaluation of the two competing linguistic components. After the evaluation no major work was done on the grammar anymore, but small improvements were made every now and then. The fourth line presents the accuracy scores for the same method about one year later. Around that time I started doing the experiments that I will describe in chapter 5. In those experiments I try to improve on this figure. The last line gives the results for the method that does not use the results of parsing. I will also compare my results to this figure later.

2.7 Ambiguity in wordgraphs

One of the major problems for most language processing systems is how to deal with the inherent ambiguity of natural language. The problem was introduced already in the first chapter. There I have discussed some of the common instances of the problem that must be dealt with in most real-world systems. Even though some ambiguities are far fetched and are merely an academic challenge, others pop up in very common situations. In this project the focus is on an application for a particular and relatively small language domain. This has the advantage that some of the potentially problematic cases of ambiguity are solved by the restrictions imposed by the application. Even though this is not a satisfactory solution in general, it does mean that for this project it is possible to focus on different aspects of the ambiguity problem.

Word sense ambiguities are often solved because of the fact that some of the meanings given by a general lexicon for Dutch, are just not included in the

lexicon for the application. The Dutch word *aankomen*, for example, can either mean *to arrive* or *to gain weight*. It will be clear that the latter sense is not appropriate for a system that can answer questions about the railway schedule and therefore can be left out of a lexicon for this particular domain.

The same thing applies for many *attachment ambiguities*. For example the sentence:

(2.24) Ik wil morgen met de eerste trein vanuit Groningen naar Amsterdam reizen
I want to travel tomorrow with the first train from Groningen to Amsterdam.

It is safe to assume that not only in this particular case, but also in other similar cases (e.g. cases where the preposition 'from' or 'to' is followed by a city name) the PP (*'from Assen*) modifies the verb (*'to travel'*) and not the noun (*train*). This means that in many cases the ambiguity can be avoided by 'hard coding' preference for the desired construction in the grammar. Again, this is not the solution of choice when a proper description of Dutch grammatical constructions is the primary goal, but it is a satisfactory solution for this application. A second, and cleaner solution would be to express a preference for the desired interpretation in the (highly domain-specific) translation of *Quasi-Logical Forms* (QLF's) into updates.

In section 2.4.7 I motivated the choice of QLF's for semantic representation by pointing at another potential source of ambiguity: quantifier scope. Originally, underspecification of quantifier scope was intended to postpone the calculation of the prefered order of the quantifiers until enough information is available. However, in many cases, again for this particular application, it is not necessary to resolve the ordering of the quantifiers at all, because of the fact that only one of the possible interpretations makes sense in this domain. In these cases, the unresolved QLF can then be translated into the correct preferred domain specific reading by adding the right qlf-to-update translation rules.

Last but not least, after the first corpora of transcribed dialogue became available, it became clear that most user utterances are so short that ambiguity is rarely a problem (in analyzing the actually uttered sentence, that is). In the previous section it was shown that if the actually uttered sentence is available, our OVIS2 language component scores a concept accuracy of 95%. Since this remaining 5% error would be difficult to eliminate (many of these utterances are ungrammatical anyway) there was very little to gain by only improving the grammar. It was also shown that when the wordgraphs were used instead of the actual utterances, the accuracy dropped considerably. Although it is not often the case that a particular hypothesis chosen from the wordgraph given by the speech recognizer needs to be disambiguated, interpreting the wordgraph introduces a new kind of indeterminacy. Even though this indeterminacy is not ambiguity in the linguistic sense, it does involve choosing between alternative interpretations of the same acoustic signal.

This interpretation ambiguity of the acoustic signal is translated into the problem of finding the optimal path in a wordgraph. An optimal path here means a path that denotes a word string that is as close as possible to the user utterance. Earlier in this chapter this problem and the approach to tackle it is introduced in some detail. The evaluation showed that although the speech recognition is still far from perfect (table 2.2 shows us that even when we have an oracle at our disposal to find the optimal path in the wordgraph, the word accuracy would be only slightly higher than 90%), the gap between what the current implementation achieved and what is possible in the ideal case (using an oracle) is still considerable. I have looked at those cases where there the update suggested by the OVIS2 grammar deviated from the update suggested by the annotators. In these cases I looked at the wordgraph and tried to find alternative paths in the wordgraph that were closer to the user utterance. This informal evaluation taught me that in quite a few cases a better hypothesis was available in the wordgraph. An example is given in figure 2.12.

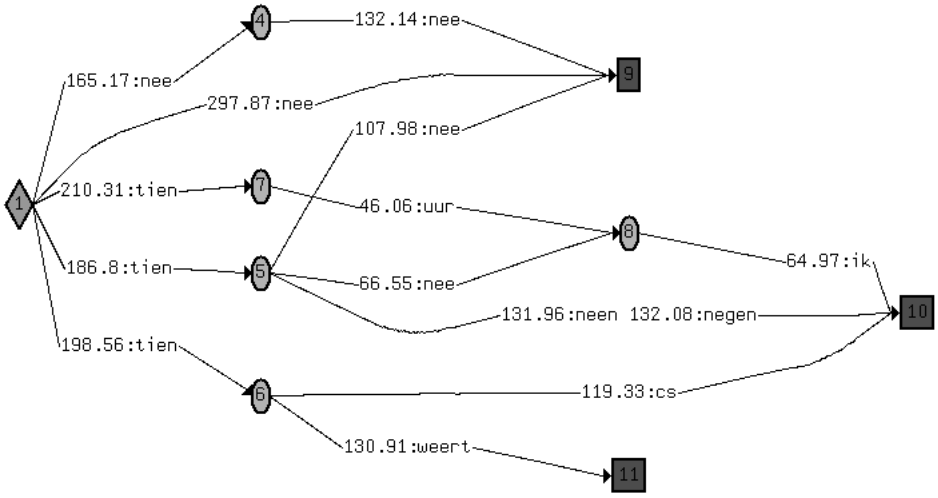

Figure 2.12: Optimal hypothesis is not found. The actually uttered sentence was *tien uur* ('ten o'clock'), but the preffered path is *Nee, nee* ('No, no').

In this particular wordgraph the user utterance *tien uur* is available in the wordgraph, but the string is not covered by a path from the start state to a final state. In order to end up with the desired hypothesis, the path denoting the string *tien uur ik* ('ten o'clock I') must be chosen. The reason for the last word (*ik*) is probably something like background noise (it could have been an artefact of spoken language, but it was not observed by the annotater and therefore most likely not there). It is clear that it is impossible to construct a single analysis for the string *tien uur ik*. The parser will analyse the first part of the string (*tien uur*) as a temporal NP and the second part (*ik*) as a pronoun. Temporal NP's can get the status of top categories, pronouns can not. Therefore, the path denoting *tien uur ik* can only be accepted if only the second part (the pronoun *ik*) is skipped. As explained in section 2.5.3, it is possible to accept a path

even when some information is skipped, but there is a preference for complete paths. If we want to choose the incomplete path, we will be punished for every skipped edge. The search algorithm preferred the path *'Nee, nee'*. *'Nee'* can get the status of a top category, which means that this path can be analyzed as a series of two top categories, without skipping any information.

The question now is what information is needed to find the prefered hypothesis. Is it possible to find clues for the better hypothesis inside the wordgraph or is external information needed? Does external information mean 'general' knowledge of the world or will it be possible to find more clues *outside* the current utterance, but within the current dialogue?

Even though the evaluation of those cases where something went wrong was pretty informal, it became clear that in a considerable number of cases the preferred hypothesis did not make any sense in the state the dialogue was in at that moment. For example, in the case of the wordgraph in figure 2.12, the (system) question that preceded this (user) utterance was about *when* the user wanted to travel. It is of course still possible that the user is so fed up with the system misunderstanding everything he says, that he just utters 'No, no' in a state of despair, but we know that that is not necessary in normal situations and would like to consider other alternatives if they are available. In this particular case, there is an alternative. One of the paths starts with *Tien uur*. As I have explained above, the problem with this path is that it is not yet in a final state and therefore this hypothesis will not receive a single parse from start state to final state. Apparently, the evidence from the *acoustic* and the *ngram* knowledge sources are not strong enough to suggest the preferred hypothesis. In quite a few cases of the informal evaluation a path was available in the wordgraph denoting an hypothesis that was closer to the user utterance than the chosen path. In some cases it was not clear what extra information would help to favour the better path over the chosen one. Some wordgraphs seemed to be such a mess that it is quite difficult to pick out the right words, in other graphs there was the choice between two (semantically) similar hypotheses. For example, I think that the choice between *Ik wil naar Haren* and *Ik wil naar Baarn* (I want to go to Haren/Baarn') can only be made on basis of accoustic evidence.

However, I could identify a considerable number of cases where a better hypothesis was available in the wordgraph and where I had the impression that knowledge of the dialogue state could help favour the desired path over others. The second observation I made was that mostly the disambiguating information was available in the system question that preceded the user utterance. In most of the cases the knowledge that in the system question the user was asked to supply information about, for example, the departure time, seemed to be a good clue to favour an hypothesis that supplied this information.

2.7.1 Related work and some possible approaches

Specialized *n*-gram models

I have suggested earlier that in the example given in figure 2.12 the evidence given by the accoustic and the *n*-gram models are not strong enough to favour the preferred hypothesis. In the current set-up we have to work with the word-graph given by the speech recognizer and we are not in the position to alter the accoustic model. The used *n*-gram model, however, is trained on the whole corpus of user utterances. The research hypothesis is that information suplied by the previous system question can be used as evidence. An obvious method that might be used to reach this goal is *to train different trigram models for every question type* (see for related work **?**)). For this particular application this is feasible, because the number of different types of questions is limited. I have identified five different types:

- Wh-locative questions (*'From which station do you want to travel?'*)
- Wh-temporal questions (*'When do you want to travel from A to B?'*)
- 'regular' yes-no questions (*'Do you want another connection?'*)
- 'confirmation' yes-no questions (*So, you want to travel tomorrow at five o'clock?*)
- a rest category (e.g. *Could you please repeat what you have said?*

The obvious disadvantage of such a method is the undesirable consequence that you have less data at your disposal to train your language model. A second disadvantage is very limited flexibility in adapting your model. In this thesis I will only consider 5 different dialogue contexts, which means that data sparseness will not be too much of a problem. A more sophisticated notion of context would split up the available data in many more parts, which is not desirable in the general case.

Interpolated *n*-gram and context models

A second obvious approach is to define a traditional trigram model and a context model and to combine these models by *interpolating* them. Interpolation is succesfully used for combining knowledge sources. The Estimation Maximisation algorithm can be used to choose the weights for the various language models in such a way that perplexity is minimal (Clarkson and Rosenfeld, 1997). However, the weakness of interpolating models is the fact that you can't account for interaction between the different knowledge sources.

Reordering of n-best solutions

A third possible approach is to produce the n-best solutions given the acoustic score and re-rank the solutions using a number of knowledge sources as proposed in **?**). In the experiments reported on in that paper a number of different knowledge sources are used to reorder an *n*-best list of hypotheses

given by the speech recognizer. Most of the knowledge sources they use are also used in the current OVIS nlp module (like recognizer scores, n-grams and linguistic coverage). Even though in the experiments they describe no dialogue context knowledge source is used, there is no reason why this can not be done. They provide an algorithm to automatically obtain the weights for the different knowledge sources. It might sometimes be an advantage to have concluded the analysis of the whole utterance, before deciding which hypothesis to prefer. However, with this approach it is again a problem to account for interaction between the knowledge sources.

Dialogue act modeling

Very relevant to the approach in this thesis is the work presented in (**?**). Although the focus in that paper is on modeling dialogue acts in conversational speech, they too, try to improve the analysis of speech recognizer output by looking at the dialogue state. The fact that in this thesis a limited number of dialogue acts is used and, moreover, that the dialogue act of the utterance preceding the user utterance (in OVIS the system question) is always given, makes the set-up used for this thesis a special case of the more general situation decsribed by Stolcke et al. They address important issues that need to be solved in order to be able to use the approach proposed here in dialogue systems that are less constrained than the OVIS system. The fact that they too observe a decrease in word error rate is promising for wider application of the approach proposed here.

The approaches pursued in this thesis

As a matter of fact I have experimented with the first two approaches mentioned above. Although I found that perplexity was reduced in both cases (compared with the baseline trigram model), the reduction was slightly higher for the MaxEnt model I discuss in chapter 5. The reason that I do not want to elaborate too much on this point is the fact that I think that there are other reasons why the approaches suggested in this thesis are worth considering to solve the problem.

In the following chapters I will investigate two ways of exploiting knowledge about the dialogue state to improve the accuracy of the OVIS2 parser. I will investigate how contextual knowledge can be added to the knowledge sources that were used so far to find the best hypothesis.

Because I think that most of the knowledge can be found in the preceding system question, I will first limit the dialogue context to just the previous question. I admit that there are cases (for example, the 'desperate user scenario' I sketched above) where more dialogue history can be very informative, but before I will consider more history, I will first concentrate on this easily accessible piece of information.

There is at least one further reason to focus on the potential contribution of context to reducing the indeterminacy in interpretation, namely the scientific question of the extent to which context contributes to understanding. As I will develop below, the context in the OVIS application is restrictive enough that it is to be expected that the contribution in this case is optimal.

In the next chapter I will first investigate a knowledge-based approach to incorporate dialogue context. I will adopt a method to describe the OVIS dialogues, explain how predictions can be made about the next dialogue move and how these predictions can be used to block certain hypotheses that are available in the wordgraph. Even though some experiments taught me that I could benefit from this approach, I will conclude that I think the method is too rigid and suggest that a way of *ranking* the different hypotheses would be preferable. In chapter 4 I will introduce a statistical framework that I will apply in chapter 5.